

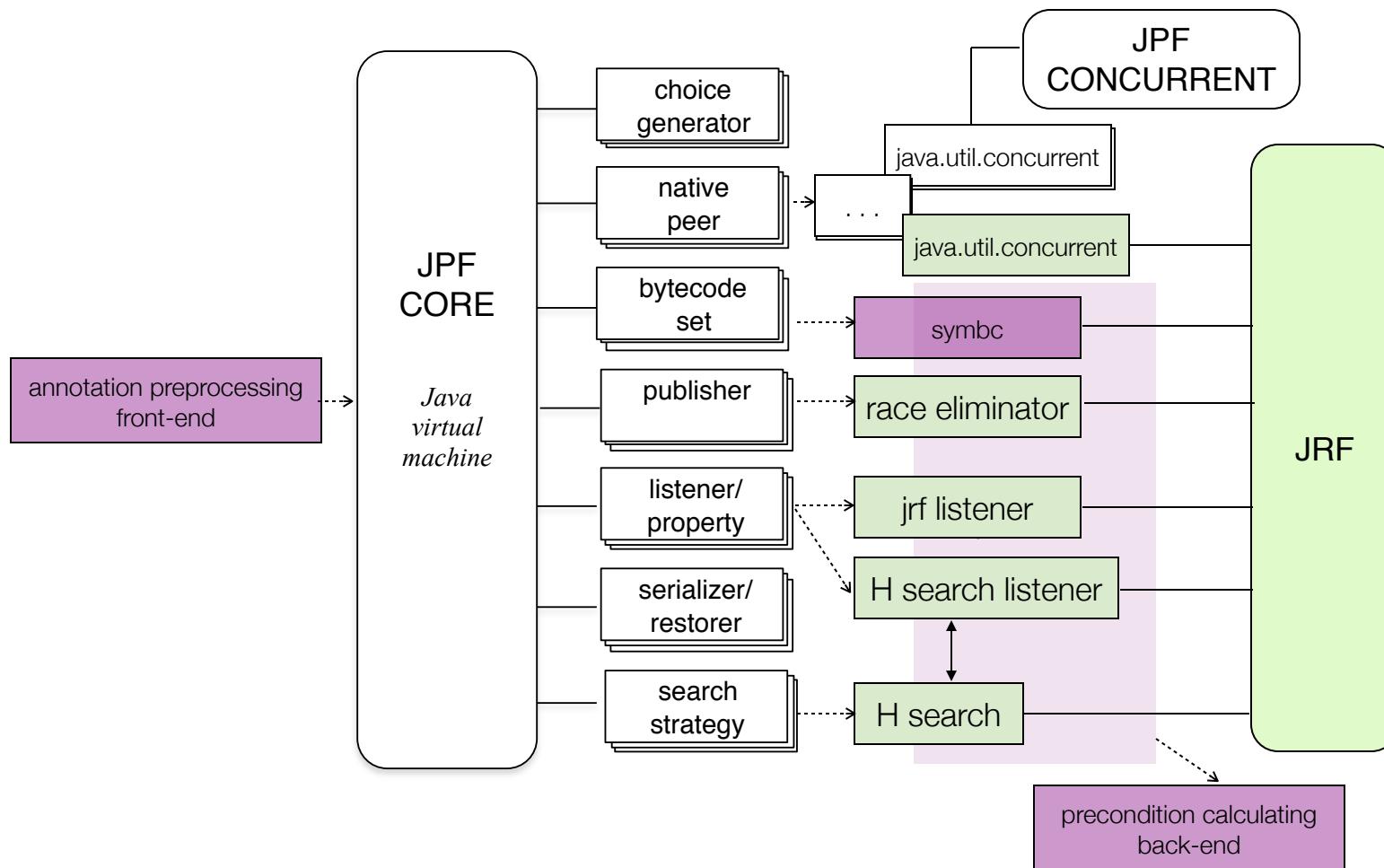
# Java RaceFinder

---

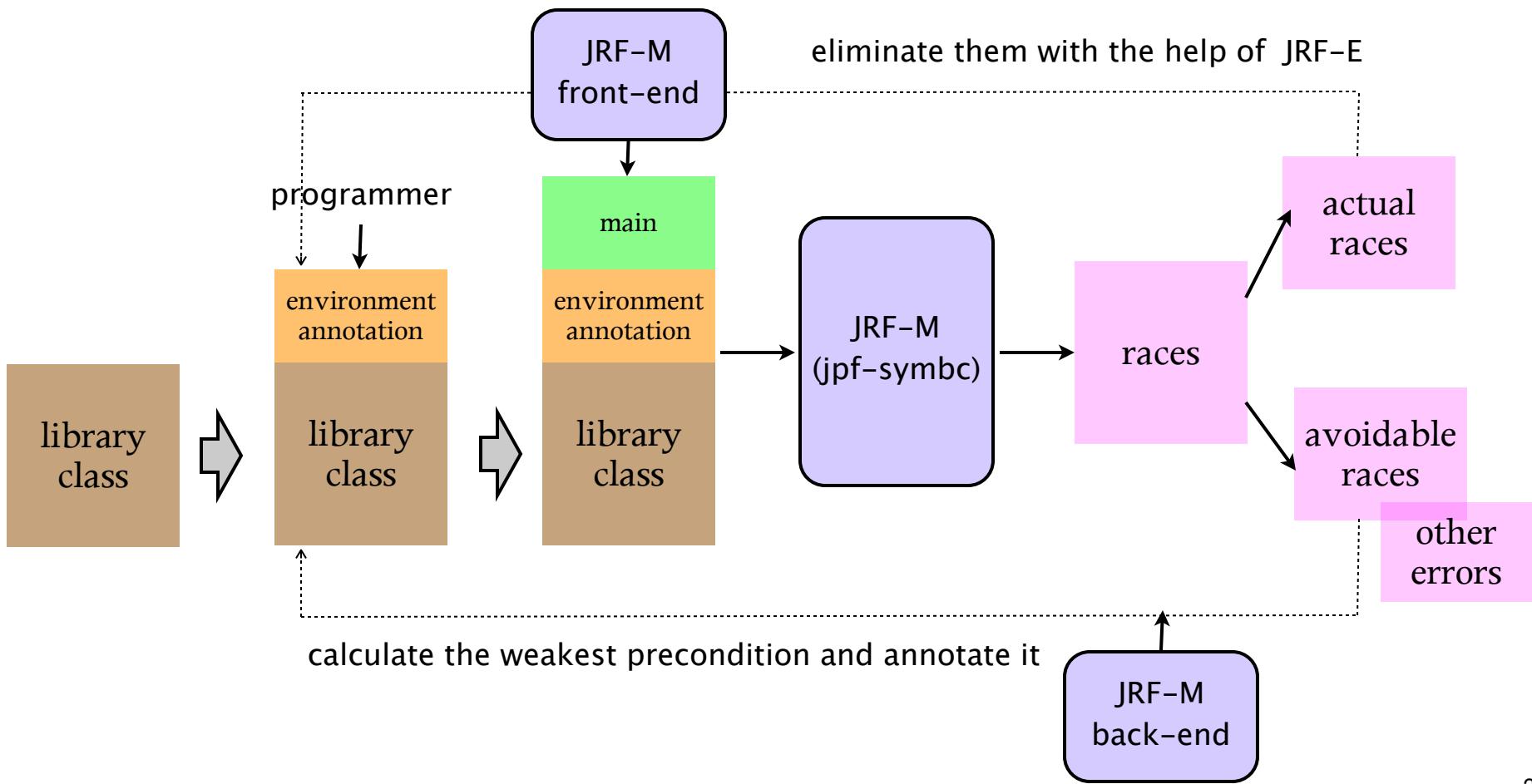
JRF-Modular as Summer Work

**By KyungHee Kim  
Department of Computer &  
Information Science & Engineering  
University of Florida**

# JRF-M : modular race detection using symbolic execution



# JRF-M operational model



# JRF-M example

library  
class

```
public class Queue<T> {
    T items[];
    int head, size;
    int capacity;

    public Queue(int capacity) {
        items = (T[]) new Object[capacity];
        head = size = 0; this.capacity = capacity;
    }

    public synchronized T deq() {
        while (size == 0) {
            try {
                wait();
            } catch (InterruptedException ex) {
            }
        }
        notifyAll();
        size--;
        return items[head++];
    }

    public synchronized void enq(T x) {
        while (size == capacity) {
            try {
                wait();
            } catch (InterruptedException ex) {
            }
        }
        notifyAll();
        items[(head + size) % capacity] = x;
        size++;
    }

    public int size() {
        return size;
    }
}
```

# JRF-M example

environment annotation

library class

```
public class Queue<T> {  
    T items[];  
    int head, size;  
    int capacity;  
  
    public Queue(int capacity) {  
        items = (T[]) new Object[capacity];  
        head = size = 0; this.capacity = capacity;  
    }  
  
    public synchronized T deq() {  
        while (size == 0) {  
            try {  
                wait();  
            } catch (InterruptedException ex) {  
            }  
        }  
        notifyAll();  
        size--;  
        return items[head++];  
    }  
  
    public synchronized void enq(T x) {  
        while (size == capacity) {  
            try {  
                wait();  
            } catch (InterruptedException ex) {  
            }  
        }  
        notifyAll();  
        items[(head + size) % capacity] = x;  
        size++;  
    }  
  
    public int size() {  
        return size;  
    }  
}
```

@ID : 0  
@Threads : 1  
@Iteration : 1

@ID : 1  
@Threads : 1+  
@Iteration : 1+  
@Preceded by 0

..

@ID : 2  
@Threads : 1+  
@Iteration : 1+  
@Preceded by 0

@ID : 3  
@Threads : 1+  
@Iteration : 1+  
@Preceded by 0

# JRF-M example

environment annotation

library class

```
public class Queue<T> {  
    T items[];  
    int head, size;  
    int capacity;  
  
    public Queue(int capacity) {  
        items = (T[]) new Object[capacity];  
        head = size = 0; this.capacity = capacity;  
    }  
  
    public synchronized T deq() {  
        while (size == 0) {  
            try {  
                wait();  
            } catch (InterruptedException ex) {}  
        }  
        notifyAll();  
        size--;  
        return items[head++];  
    }  
  
    public synchronized void enq(T x) {  
        while (size == capacity) {  
            try {  
                wait();  
            } catch (InterruptedException ex) {}  
        }  
        notifyAll();  
        items[(head + size) % capacity] = x;  
        size++;  
    }  
  
    public int size() {  
        return size;  
    }  
}
```

@ID : 0  
@Threads : 1  
@Iteration : 1

@ID : 1  
@Threads : 1+  
@Iteration : 1+  
@Preceded by 0

@ID : 2  
@Threads : 1+  
@Iteration : 1+  
@Preceded by 0

@ID : 3  
@Threads : 1+  
@Iteration : 1+  
@Preceded by 0

JRF-M  
front-end

Group1 : 1 thread, 1 iteration  
Queue(int capacity)

Group2 : 1+ thread, 1+ iteration  
T deq()  
void enq(T x)  
int size()

2 symbolic parameter

1 object return value

# JRF-M example

environment annotation

library class

main

```
public class JRFMQueueTest {
    public static void main(String[] args)
    {
        JRFMQueueTest test = new JRFMQueueTest();
        test.doTest();
    }
```

```
    Queue<Integer> obj;
    int threadGroup0=1;
    int threadGroup1=2;
    int iterationGroup1=1;

    @Symbolic("true")
    int sym1=0;

    @Symbolic("true")
    int sym2=0;
```

```
    void doTest() {
        testGroup0();
        testGroup1();
    }
```

```
    void testGroup0()
    {
        for ( int i=0 ; i < threadGroup0 ; ++i)
            new Group0Thread().start();
    }
```

```
    void testGroup1()
    {
        for ( int i=0 ; i < threadGroup1 ; ++i)
            new Group1Thread().start();
    }
```

```
class Group0Thread extends Thread {
    public void run()
    {
        test0();
    }
}
```

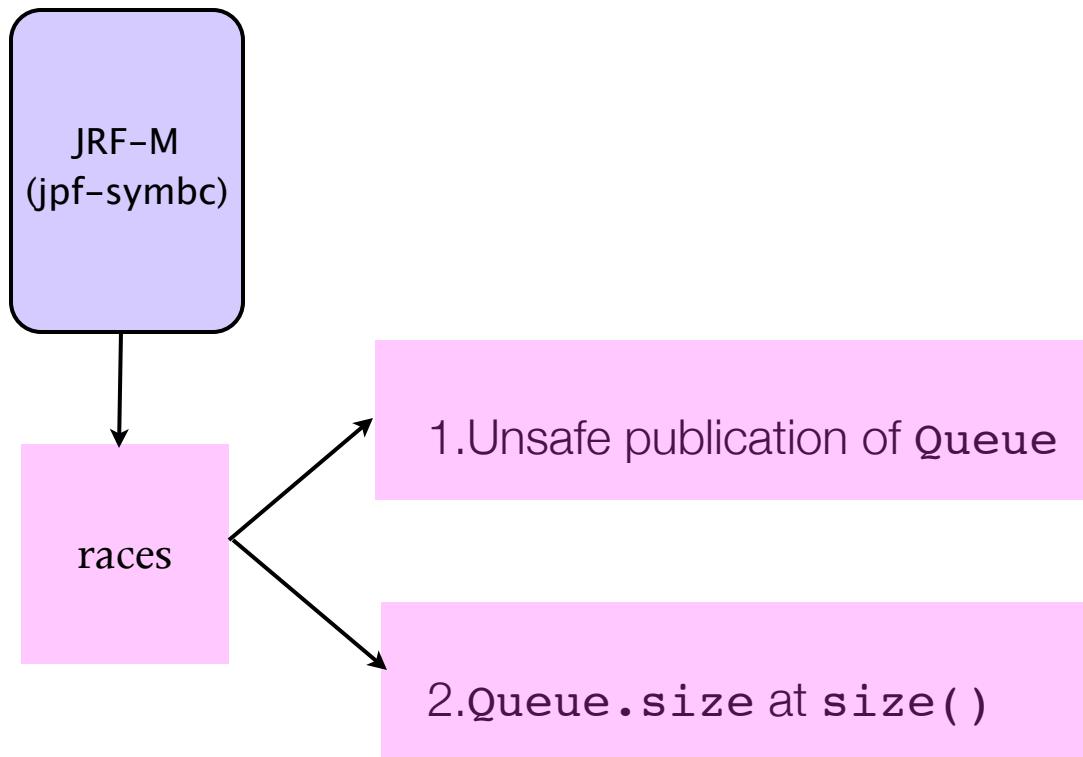
```
class Group1Thread extends Thread {
    public void run()
    {
        for ( int i=0 ; i < iterationGroup1 ; ++i ) {
            int option = gov.nasa.jpf.jvm.Verify.getInt(1, 3);
            if ( option == 1 ) test1();
            else if ( option == 2 ) test2();
            else test3();
        }
    }
}
```

```
    void test0()
    {
        obj = new Queue<Integer>(sym1);
    }
```

```
    void test1()
    {
        Integer t = obj.deq();
        t.intValue(); // check if returned object's public methods
    }
    void test2()
    {
        obj.enq(sym2);
    }
    void test3()
    {
        obj.size();
    }
```

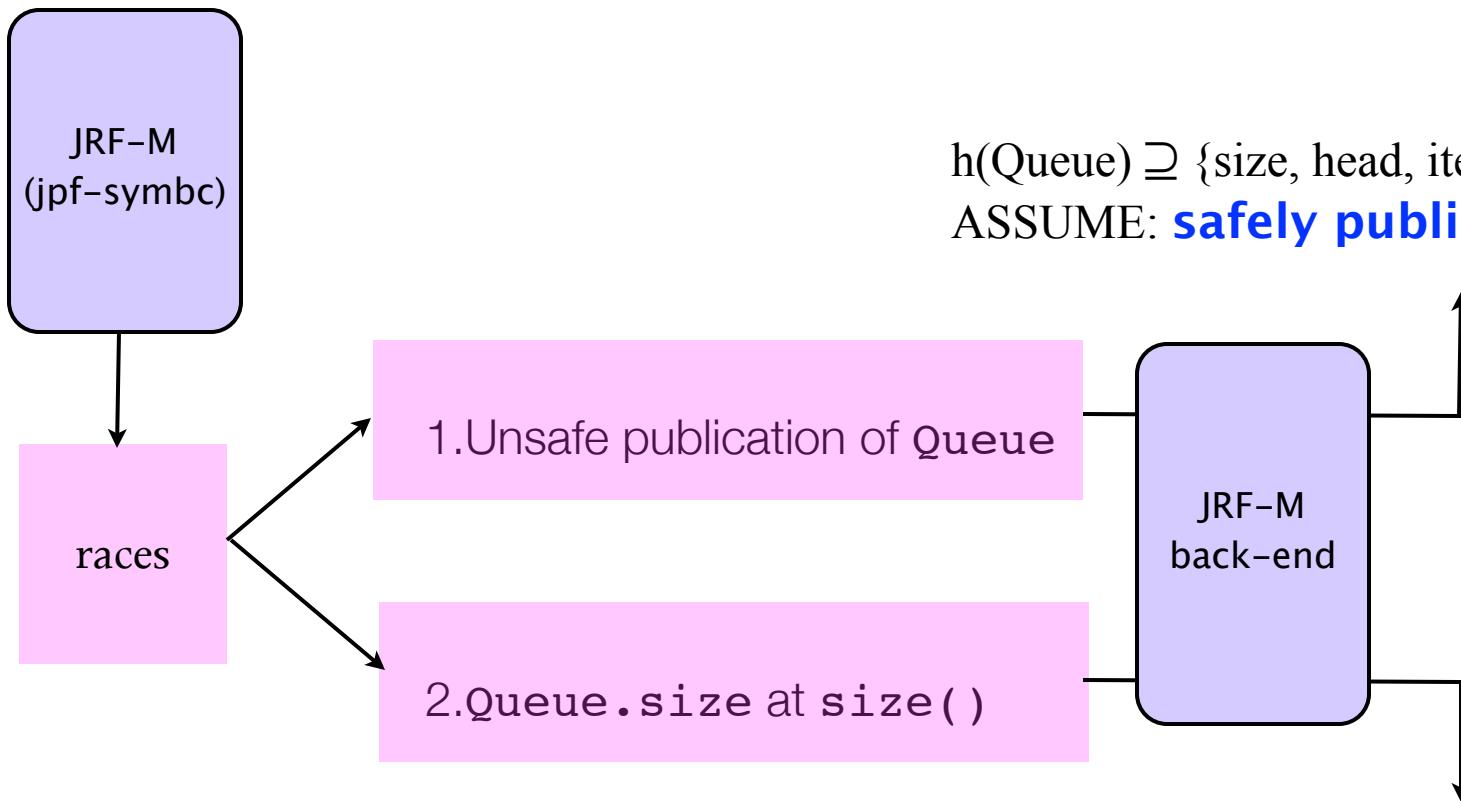
# JRF-M example

---



- deadlock at `deq()`

# JRF-M example



$h(\text{Queue}) \supseteq \{\text{size}, \text{head}, \text{items}, \text{items}[]\}$

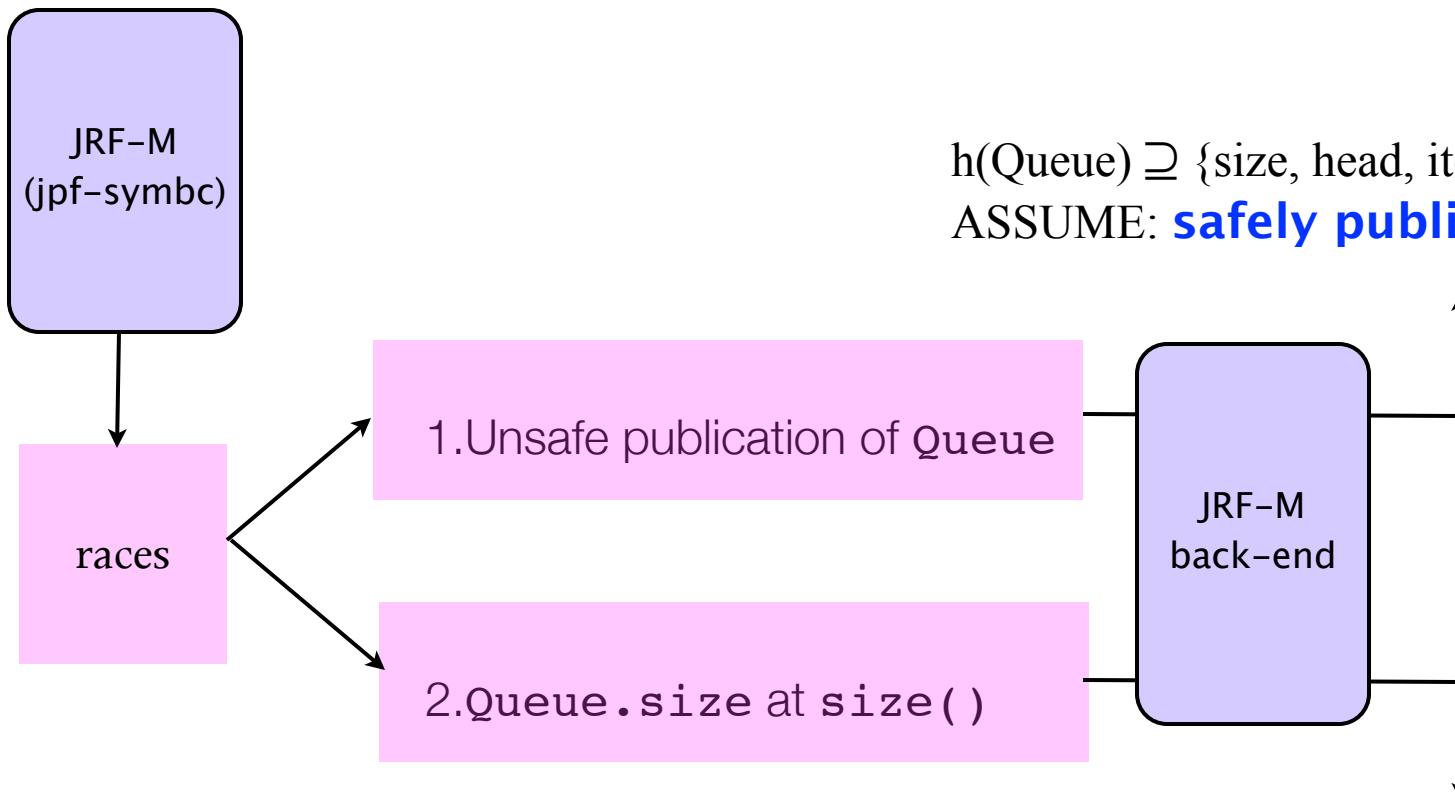
ASSUME: **safely published this Queue**

$h(\text{CurrentThread}) \supseteq \{\text{size}\}$

ASSUME: **lock this Queue before**

- deadlock at `deq()`

# JRF-M example



$h(\text{Queue}) \supseteq \{\text{size}, \text{head}, \text{items}, \text{items}[]\}$

ASSUME: **safely published this Queue**

$h(\text{CurrentThread}) \supseteq \{\text{size}\}$

ASSUME: **lock this Queue before**

● deadlock at `deq()`

→ add more annotation to `deq()`  
such as `@Iteration : less than or equal to 2.Iteration`

# Discussion

---

- public fields cannot be proven to be race-freedom
  - Can we annotate the protection mechanism used in the class code?  
(ex. a protecting lock or read other volatile before...)
- static fields and static methods should be considered also
- which format is good for specifying the weakest precondition?
- better optimized way to organizing main stub in order to reduce search space

# Summer Works Schedule

---

- **5/29 – 6/19** Design of overall approach & Preliminary design document ([2weeks](#))
  - define the interface/structure of the annotation to each model class to augment the environment information
  - define the structure of the precondition for a data race freedom
  - search for appropriate test suites
- **6/20 – 7/17** Coding & Testing ([4weeks](#))
  - parse & analyze the model class annotation about the environment
  - generate the entry stub (main routine) using (1)
  - calculate summary function  $h$  during a symbolic execution of (2) using JPF
  - compute a race free precondition for each class from the result of (3) and save as an additional annotation
- **7/18 – 8/7** Applying to various test suites and modify the design document & codes as necessary ([3weeks](#))
- **8/8 – 8/16** Final document ([1week](#))

---

Thank you!